Escuela
Politécnica
Superior

Application for delegated control of connections in a computer laboratory

Degree in Computer Engineering

Bachelor's Thesis

Author:

Rafael Olid Rodríguez

Tutors:

JUAN ANTONIO GIL MARTINEZ-ABARCA

MARCELO SAVAL CALVO

Universitat d'Alacant
Universidad de Alicante

Acknowledgements

I would like to express my gratitude to my tutors Marcelo and Juan Antonio. Thank you for sharing your knowledge, for the commitment and for your availability.

I would also like to thanks my family, specially to my parents and my sister, who have been always supporting me.

Thank you to all my colleagues, not only my classmates, but also my laboratory partners.
And lastly thanks to Maria, no matter where you are, you always have a piece of advice for me.

To all these people mentioned before, not only this project, but the entire degree, would have been impossible without your support.

To all my family, specially my parents and my sister,
without them all of this would not be possible.

As a network administrator, I can take down the network with one keystroke. It's just like being a doctor but without getting gooky stuff on my paws.

Scott Adams.

Abstract

Currently, the Polytechnic School of the University of Alicante has a tool through which they make network cuts in specific computer laboratories to perform exams without students using resources that can be found on the Internet. However, this system has two problems: the first is that it is becoming obsolete, as the mechanisms it uses were designed for specific network devices and can no longer be renewed. The second is the level at which filters are applied, as it currently only regulates connections to the outside network, but does not allow limiting connectivity between devices within laboratories. The objective of this project is the design and implementation of a new system that allows to eliminate the dependence of network devices working at node level, which allows to increase the offered functionalities. This is possible because it will change the level where the filters are applied, instead of doing it in the router as the previous system, the filters are established in the nodes, which allows isolating a node if necessary. In addition, the proposal made in this project, leaves open the possibility of implementing system filters, offering the possibility of further limiting the tools that a student can access while the filters are active.

For the development, Ansible has been used as an automation technology. By using Ansible, the filters desired by the teacher or system administrators can be deployed in the selected laboratories automatically and easily.

In order to validate the implemented system, different tests have been carried out that expose different real situations in which the system will be applied once deployed. This allows observing the response of the system in normal regime, and before possible failures that may occur, whether caused by the same system or by a student from his node. With the validation, the correct functioning has been verified for the different cases.

Resumen

Actualmente, la Escuela Politécnica Superior de la Universidad de Alicante dispone de una herramienta mediante la cual efectúan cortes de red en laboratorios de ordenadores determinados para poder realizar exámenes sin que los alumnos usen recursos que pueden encontrar en Internet. Sin embargo, este sistema tiene dos problemas: el primero es que este se está quedando obsoleto, ya que los mecanismos que usa fueron diseñados específicamente para unos dispositivos de red concretos y ya no se pueden renovar. El segundo es el nivel en el que se aplican los filtros, ya que actualmente sólo regula las conexiones con la red exterior, pero no permite limitar la conectividad entre dispositivos dentro los laboratorios.

Este proyecto tiene como objetivo el diseño y la implementación de un nuevo sistema que permita eliminar la dependencia de dispositivos de red trabajando a nivel de nodo, lo que permite incrementar las funcionalidades ofrecidas. Esto es posible debido a que se cambiará el nivel donde se aplican los filtros, en lugar de hacerlo en el router como el sistema anterior, los filtros se establecen en los nodos, lo que permite aislar un nodo si fuera necesario. Además, la propuesta que se hace en este proyecto deja abierta la posibilidad de implementar filtros de sistema, ofreciendo así la posibilidad de limitar aún más las herramientas a las que un alumno puede acceder mientras los filtros se encuentran activos.

Para el desarrollo, se ha empleado Ansible como tecnología de automatización. Mediante el uso de Ansible, se podrán desplegar los filtros deseados por el profesor o los administradores del sistema en los laboratorios seleccionados de forma automática y sencilla.

Para validar el sistema implementado, se han llevado a cabo diferentes pruebas que exponen diferentes situaciones reales en las que se aplicará el sistema una vez desplegado. Esto permite observar la respuesta del sistema en régimen normal, y ante posibles fallos que se puedan producir, ya sean provocados por el mismo sistema o por algún alumno desde su nodo. Con la validación, se ha comprobado el correcto funcionamiento para los diferentes casos.

Índice general

Índice de figuras

List of tables

List of Codes

1. Introduction

In this first chapter of the project I will include the motivation and context, where I'll provide a general framework of this study. After that, an overview of the system will be presented. The main objectives are then stated. Last, I introduce the main tools that will be used along the project to allow the reader to have a better background of it.

1.1. Motivation and context

Every day big organizations are growing and being more automatized. Almost every employee has a computer or needs to access to some resources in the network. This drives to a more complex management of the system and the network. This project is focused on two main aspects: control the execution permits and the access of some computers in the network to some specific resources, and automatizing the work of applying those controls or limitations. This could be accomplished in two different ways: acting on the network or on each computer, as we will see later.

Specifically, this project is developed for the University of Alicante (UA). Nowadays, the UA has a large campus in which a lot of different degrees can be found. The University had in 2018 more than 25000 students, 1348 researchers and professors, and more than 500 administrative staff (*University of Alicante*, 2019). With this numbers the need of automation and control of the local network and the resources is obvious. On the other hand, the UA is divided into 7 faculties, including Polytechnics, Sciences, Health, Economics, Law, Education, and Philosophy. This project aims to be applied in the Polytechnic School (EPS, from the spanish name Escuela Politécnica Superior), where the need to limit the access to some Internet resources in specific cases, such as exams, is known. Table 1.1 shows the subjects that required from network access limitations last academic year. As it can be seen, there are already several subjects that require this type of service and it will continue growing.

The EPS is composed by four buildings, as it is shown in Table 1.2, two of them with laboratories with a variable number of computers.

In those laboratories, there are practical classes of different subjects and also exams take place there. In some cases, teachers want to limit the access to internet or some resources to avoid students from

Academic Year 2017-2018
Mathematical Foundations Of Engineering III
Metal Structures
Simulation and Optimization in Civil Engineering
Programming 3
Mathematics I
Geotechnical Engineering
Statistics
Programming Fundamentals
Electrical Technologies
Programming Challenges
Programming 2
Expansion Of Separation Operations
Information Systems

Table 1.1: Subjects that required a connection cut during last academic year.

EPS Buildings	Usage
EPS1	Laboratory classrooms
EPS2	Offices
EPS3	Offices
EPS4	Laboratory classrooms and offices

Table 1.2: Buildings that compose the EPS.

cheating or downloading banned information. However, the teacher might want to allow some specific access, for example servers where the exam is evaluated. In order to do this, the EPS currently working with a tool that allows the professors to ask for a cut in the network of the lab or look up if there is already one. As it is shown in Figure 1.1, the application allows the teacher to look up if a connection cut has been implemented or not. Furthermore, it could be interesting to be able to limit some other execution permits, for instance, the mounting of external USB memories or the execution of several applications. Nevertheless, this is not possible in the current system.

Figure 1.1: Query for connection cut.

In Figure 1.2 we can observe the interface that a professor will see to ask for a connection cut. In it, the user can specify the exact date and time of the exam, their personal data (so the administrators are able to contact them if there is any inconvenience), the subject of the exam and the laboratories in which it will take place.

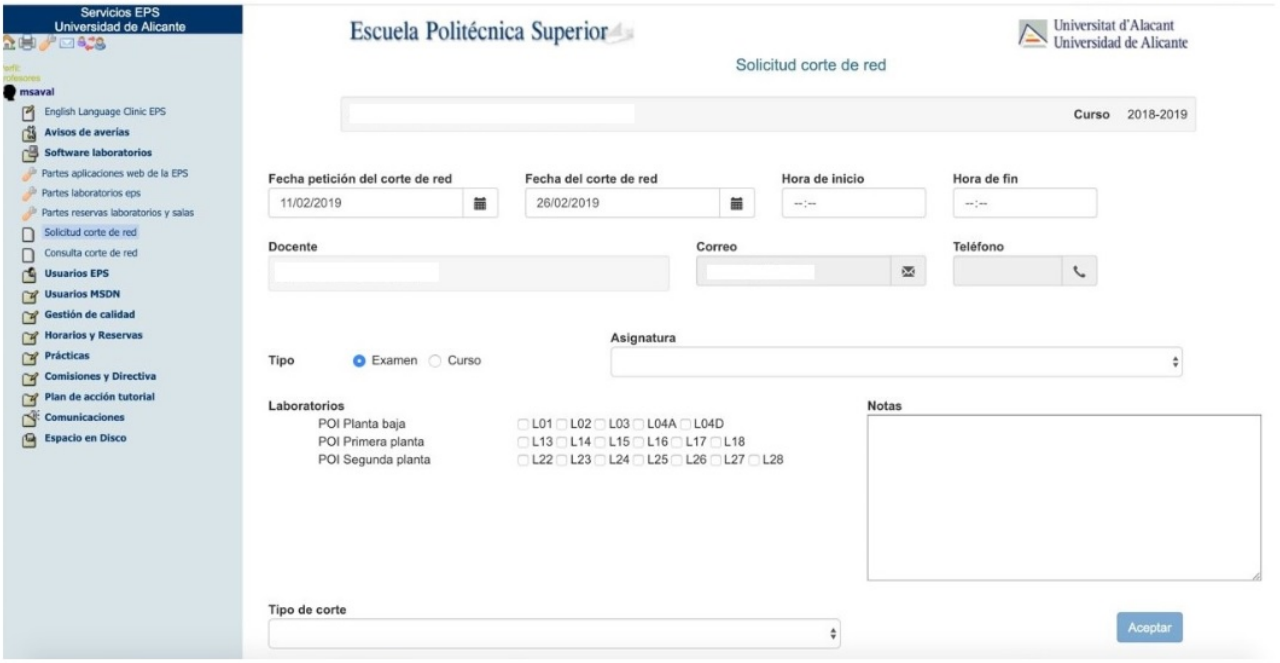

Figure 1.2: Form for a professor to perform a connection cut in a laboratory.

Figure 1.2 is the form that a professor sees if he or she wants to perform a connection cut to a specific number of laboratories. The current system has two main problems and both are originated because the system is hardware-dependent. The first problem is the obsolescence of the hardware, in this case the routers. The application was developed specifically for the routers in the EPS1 building but, with the acquisition of new hardware, the application could not be used anywhere else. The second problem arises because the filters are applied in the router as it will be explained in the next paragraph, and not in the nodes from the laboratories or a switch between the router and the laboratories. This two problems arose the need of implementing new tools. As it can be observed in Figure 1.3, the current system is able to deny the connection from the nodes to any external server, but it cannot block the connection between nodes from the same laboratory. It is also important to notice that the implementation of the network in the EPS provides each laboratory with a different network. This means that if a student is in L01, he cannot establish a connection with another student which is in another laboratory. Furthermore, there is another network in which all the servers from the EPS are located. This last network is able to reach every laboratory in the EPS.

As it will be shown later, the application to be implemented bears a great resemblance to the one that

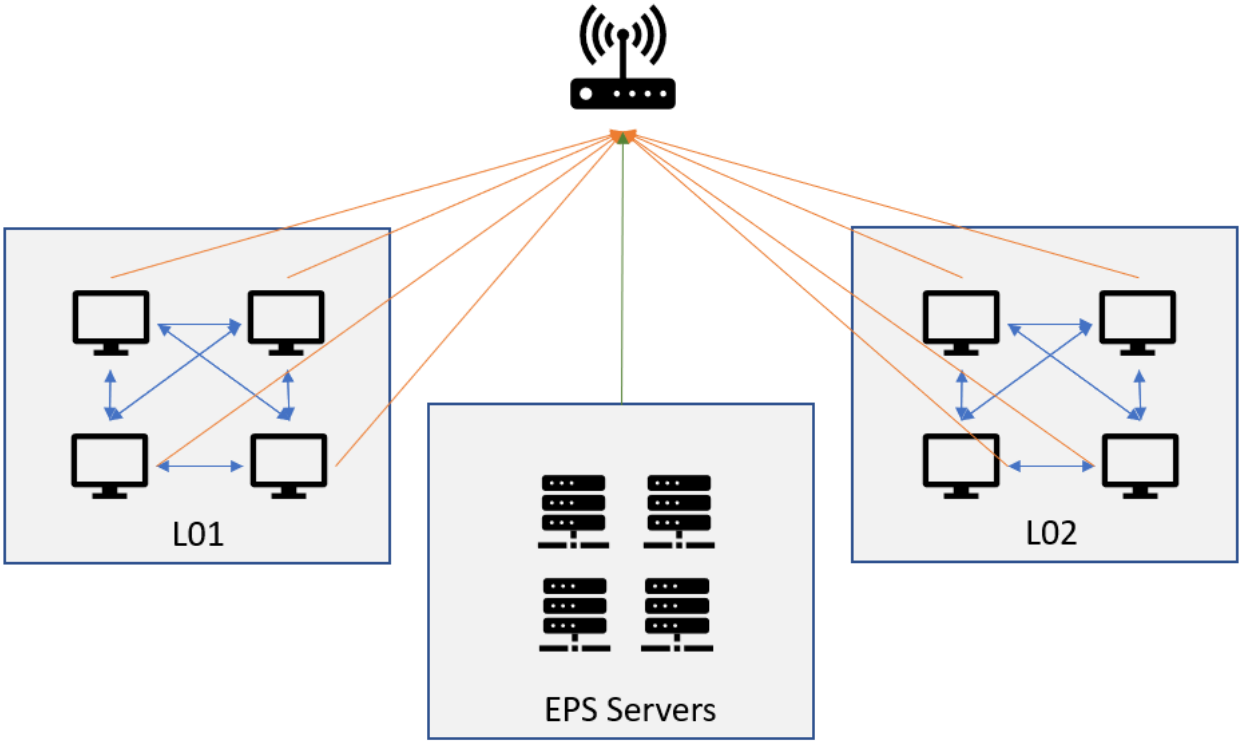

Figure 1.3: Network logical distribution.

currently exists. However, there is a huge difference between both applications. In the current one, the request of the teacher is implemented in the routers of the EPS1 in order to block the internet traffic within a laboratory network. In the application that is presented in this report, the filters are not applied in the routers itself, but to each of the computers individually, what will allow to the teacher to additionally block connectivity between the computers from the same laboratory. Furthermore, now the filters can only be applied in the building EPS1 because they are designed for the specific routers that can only be found in that building. With the new application, the hardware will not be a problem, because the application will depend on the operative system, not in the network level the filters are applied in. Even though there are some switches that provide that kind of filters, this rely on MIB objects from SNMP

1.2. Overview

Currently, computer administration includes several tasks such as user accounts managing, network maintenance, server administration, etc. Most of the tasks have been and are still performed in a traditional way with no (or very little) automation. For instance, the case of computer network permission strategies requires tedious and time consuming effort, since a set of directives have to be applied and

checked constantly to ensure no user altered the permissions. This is not the only problem that can be found in traditional computer administration, so is the dependency on the expert. As it is shown in Figure 1.4, in order to configure a computer inside a network, the system administrator has to establish a connection with the computer in question, typically with SSH (remote connection), and then apply the desired configuration through the command line or scripts. This type of administration is not only tedious and repetitive, but also relies completely on the ability of the system administrator. The fact of relying completely on the system administrator implies that only this person will be able to solve the problems that can arise, something that companies that want to provide a service 24/7 cannot afford. The introduction of automatized tasks will help minimize those problems. Moreover, tiredness can make System Administrator (SA) to fail, compromising the system.

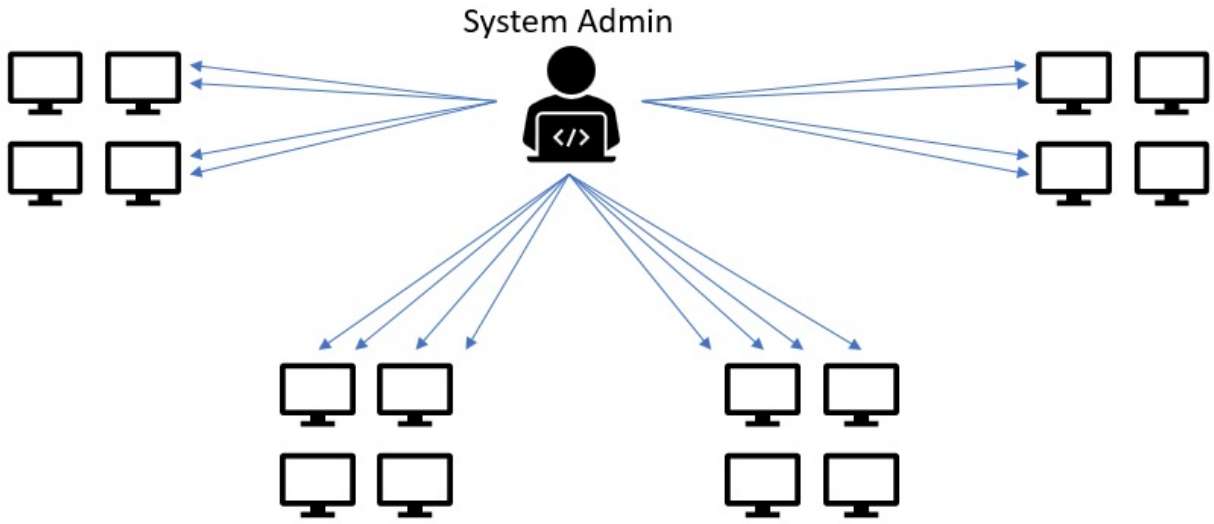

Figure 1.4: Traditional administration example.

To overcome those problems, there are alternatives to automatize traditional system administration processes. In this project, we will focus on a technique called Infrastructure as Code (IaC). IaC is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools (Keif Morris, 2016). This type of administration has five main goals:

- Ease of change. The ability of applying changes easily.
- Save time from repetitive tasks.
- Changes can be applied by someone who does not have as many knowledge of the network as the system administrator, for instance, an operator.

- Automatic recovery from errors.
- IT upgrades. Design, development, tests, monitorization, etc.

To achieve these goals, a server is necessary between the administrator and the nodes in order to be able to automate the connections and to modify any device connected to the network. As a result the scenario will be as the one shown in Figure 1.5.

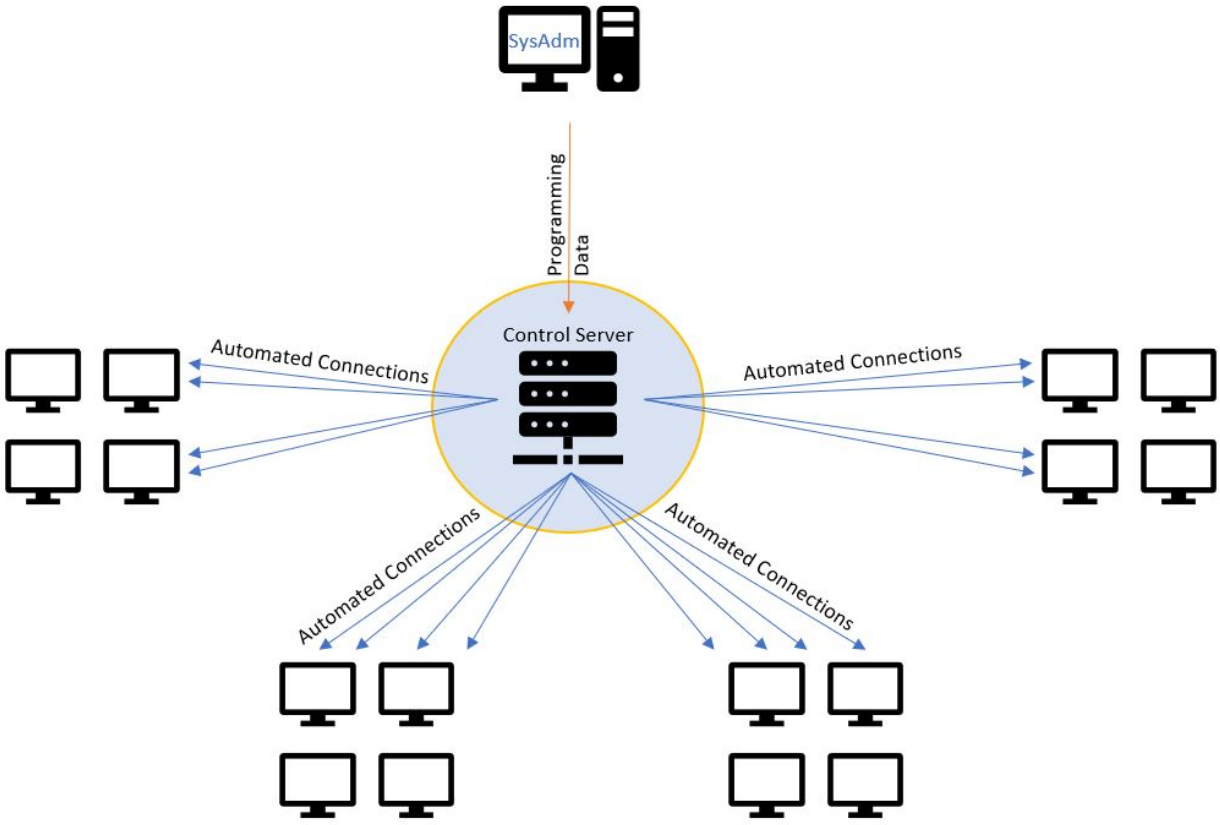

Figure 1.5: IaC administration example.

In order to operate within the network, a new configuration model as the one showed in Figure 1.6 is needed. This new model will have two administrator profiles. On the one side there will be a programmer, which is the traditional SA. In this case the SA will program the solutions in order to automatize everything as much as possible. On the other side, a new operator appears. The operator is basically someone who does not have a strong knowledge on system administration, but, using the tools programmed by the administrator, it will be able to modify the behaviour of the nodes at any moment (Bass y cols., 2015). The programs implemented by the SA will often be templates filled with variables. The operator will modify the variables to change the output, without the need of knowing what is really happening at every moment. This type of templates and programs will be discussed and

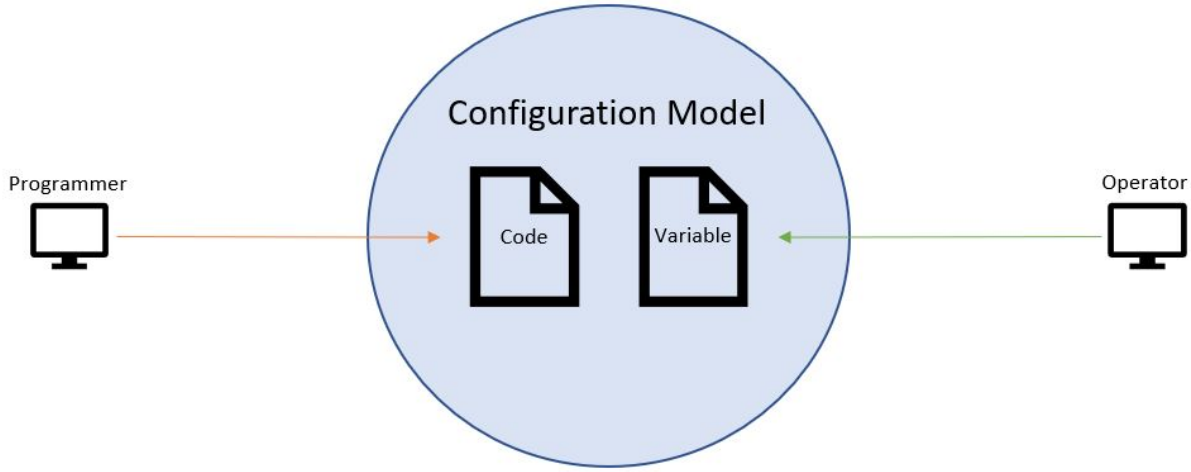

Figure 1.6: IaC configuration model.

shown in chapter 2.

Once seen the benefits of using IaC, it is necessary to analyze what problems might arise while using it. There are mainly three:

- Programming effort. As it was stated before, the users might not be experts on system administration, hence there must be an effort to make the software user-friendly, robust and transparent.
- Errors scale. The kind of system in which this is applied is usually large in terms of computers. Thus, whatever error affects many nodes. For example, if a deployment is not done with the correct parameters, the nodes could stop providing the intended service.
- Control server. The server has access to every node within the network, it is a critical point for the system from a security point of view. All the transactions in the configuration of the system are made by the control server, and if it is hacked the full system integrity is compromised.

The situation in which we are going to work, is the one we can find in the EPS of the University of Alicante. In it, some of the teachers have the necessity of controlling the access to certain websites or programs during an exam that could give students access to information that is not allowed during the test. Nonetheless, there could be the case when students need some specific or minimum access to, for example, servers in the university to upload some code or MOODLE platforms, the execution of a specific programs. Once it was determined the need to isolate the computers in the laboratories of the EPS when an exam is being carried out, a system was created so the teachers contact the SA of the EPS in order to carry out this network cut of one or more laboratories. When the request is done, the SA has to configure the laboratories networks. However, the SA cannot pre-apply the cut since

there could be other classes in that laboratory, delaying the start of the test. This tardiness increases if we think that every computer in the laboratory has to be configured independently. That is why the established solution was to act against the routers, which implies that only network filters can be applied and, if a student manages to surpass these filters, the teacher cannot know at the time of the examination. This form of access is not very complicated. The problem arrives when it comes to many computers because this task becomes repetitive. Also, despite using tools such as scripts, not all users may want the same filters, and there is always the possibility that some order fails and the execution of the script is not complete. In addition, it should be noted that if it is a non-expert user, such as a university professor, not only will be a repetitive activity, but also a complexity greater than their knowledge in the field.

1.3. Objectives

The objective of this project is to implement a system for delegated control of connections and systems in a computer laboratory. In order to accomplish this we have three sub objectives which contain the tasks that will lead to the fulfillment of the main objective:

1. Objective 1: System analysis to establish the main modules and interactions. The first objective is to obtain a system analysis where the main modules and interactions will be defined. Moreover, we define the security and fault tolerance requirements.
    - Task 1.1. System analysis. In this task I'll analyze the main modules of the system, such as the automation technology that it will use, the network environment in which is going to be developed and the security requirements that a system like this needs. Once the modules are defined, the interactions between them will be defined along with the problems or issues that should be faced in the implementation. In order to do this I will meet the head of system administration in the polytechnic school to collect all possible information.
    - Task 1.2. Proposal. After analyzing the system completely I will make a formal proposition for this project to implement the described application.
2. Objective 2: Implementation of the automation system. The second objective is to implement the entire automation system. This system will be composed by an application and the automation structure.
    - Task 2.1. Implement Application. This task consists in the implementation of an application within the server. This application will process the form fulfilled by the teacher and generate a set of files. Once the files are created it will also schedule the deployment of the filters.

- Task 2.2. Implement automation infrastructure. This task will be the implementation of the entire automation structure which will receive the files generated by the application and will be in charge of the deployment of those files through SSH connections to the nodes selected by the teacher.

3. Objective 3: Validation of the automation System
   - Task 3.1. System requirements. Analysis of the system's requirements in order to keep control of the filters that the teacher asked during the period specified.
   - Task 3.2. Application Tests. Several simulations will be run in order to check that the answer of the system is the desired one.

1.4. Proposal

With the aforementioned context and the tools described, we can see that the current system in the EPS has an obsolescence problem, and also lacks from the ability of blocking the connection between computers from the same laboratory. In this work, I propose an automated system for administration delegation to manage the access of a group of computers to network resources, similar to the one in (Pieplu, 2019), mainly focused on examination situations. Moreover, the proposal could be extended to the delegation of system filters, but this will be left for future works. Another aspect is that the current system does not alert when a user gets around the filters. Hence, in this proposal the system will alert and react when a hack occurs. Taking into account the tool that is currently used in the EPS, the aim is to migrate the idea into a new tool which is not limited by the network level in which the filters are applied, but only by the operative system of the machines that can be found within the EPS. As it can be seen in figure 1.7, when either a teacher or a SA fills the form in the application, this application will generate a configuration file in order to apply the desired filters to the nodes. Later, the automation system, with the configuration file generated by the application, will deploy the filters according to the configuration file to all the nodes specified by the teacher or SA.

According to the figure 1.7 we can distinguish three main components:

- Application (APP in the schematic). This component will be in charge of gathering the necessary information from the teacher or System Administrator. This information will be processed, and the system will generate the configuration files that the automation system uses later. In order to develop this task, the application will send the information gathered via socket to the server in which the automatization system is located.

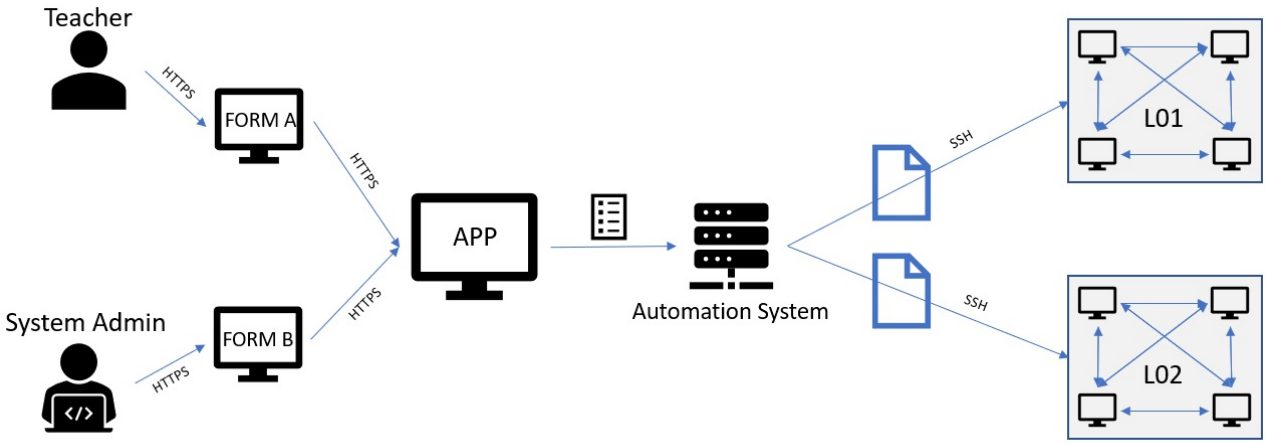

Figure 1.7: Components of the System

- Automation System. The information sent by the web application will be processed by an application located in the server, which will generate the configuration files that the automation system need in order to deploy the filters. The deployment of the system will be done using SSH as a tool to communicate with the nodes.

- Nodes. This element will receive all the information from the automation server through SSH, which will be tightened up to provide a secure connection. The filters later will modify the behaviour of this nodes through tools as iptables and AppArmor.

In order to migrate the application and improve the flexibility of the system to add changes, I will use Infrastructure as Code which allows managing IT systems with machine-readable definition files. This allows any laboratory to be administrated in any situation required by the teacher or the system administrators. In order to do this project, I will use several tools, among which we can highlight Iptables and AppArmor.

2. Implementation of the automation system

In the third chapter, I will explain the implementation of the proposed system. this includes to set an specific environment in which I can replicate the network that we can find in the EPS. After that, I will proceed to the implementation of an application which will process the data sent by the teacher, and finally the implementation of the automation system which will allow the application to schedule the execution of the tasks which will set the filters in the laboratories.

2.1. Implementation context

It is important to emphasize that this project is developed to operate under the infrastructure of the EPS. In addition, we shall clarify that in this project I face the problem of network access filtering, and leave the system filtering for extended versions. Nonetheless, it is important to notice that the proposal includes everything, the current implementation is fully compatible with extensions in system controls. For testing, a replica of the EPS system was used on an OpenStack virtual platform. This virtual environment is formed by 3 different networks, the first will be the network in which the automation server is located, the second the network belonging to the laboratory 1 and the last the laboratory 2 network. As can be seen, the structure that is followed in the virtual environment is identical, so that each laboratory has an isolated network, all visible only from the network segment where the server is located. This environment is shown in Figure 2.1.

This system will be composed by a C++ application to process the information acquired from the web interface and an automation structure that will allow us to reach all nodes in the network to be able to configure them as required by the teacher or SA. The web interface will be the same as the one that is currently used in the EPS. The C++ application will be in the automation server, and it will be specifically designed for this server.

2.2. Automation System

The automation system has two main modules. One is the application which will process the information sent from the application and it will generate the necessary files in order to configure the

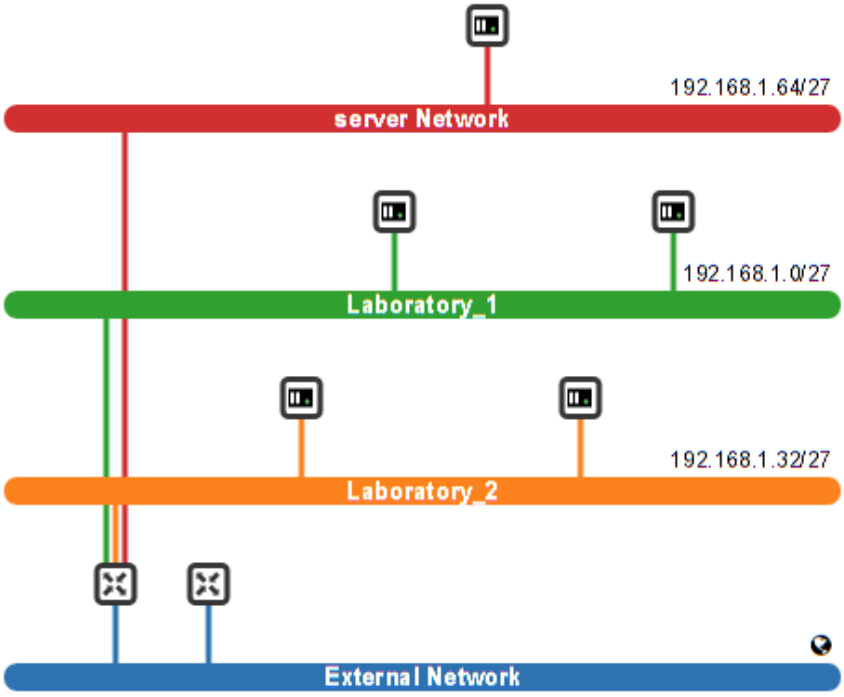

Figure 2.1: Logical distribution of the network of the project. The horizontal lines represent networks, from top to bottom it is the server network, the first and second laboratories, and last the external network (Internet, department servers...)

laboratory as the teacher wants to. The other one is the automation system. With the configured files generated by the application, the automation system will start the deployment of the filters into the nodes and keep track of them until the finish time, when it will set the filters back.

2.2.1. Ansible automation

Nowadays, there is a high amount of automation mechanisms in the market. Taking into account the objective of this project, which is a system which has the ability of alerting the user and the SA that a filter has not been applied correctly or it has been revoked, we can highlight several options between which we can find:

- Puppet. Puppet is a tool with great maturity and great stability. It should also be noted that this tool is open-source, and you can make use of its code (Alfke, 2017).
- Chef. One of the main advantages of this tool is that it is developed around Git, which makes version control very robust. However, the learning tool is too slow (Taylor y Vargo, 2013).
- Ansible. Ansible is also an Open-Source tool. It has two very important features that make it stand out from other possible tools. The first one is made in Python, which makes the learning

curve and the deployment of the application much easier. The second one doesn't require any agent, that is, with a simple SSH connection we can start working on a new master-client architecture (Shah, 2015).

After evaluating the main feature of each technology, we decided that the best tool to develop the project was Ansible because it is an open-source tool, and also because the SA from the EPS have already developed some tools with Ansible in order to perform server maintenance within the EPS, so they have some experience in it, and the application could be used in the EPS easily. Moreover, with Ansible, we will be able to reach every node or set of nodes independently of where it is in the network as said in (Alibi, 2018). Also, the same filters can be applied to a laboratory or to a set of nodes, as shown in figure 2.2, where L01 has all the computers filtered while L02 has only the top-left computer restricted.

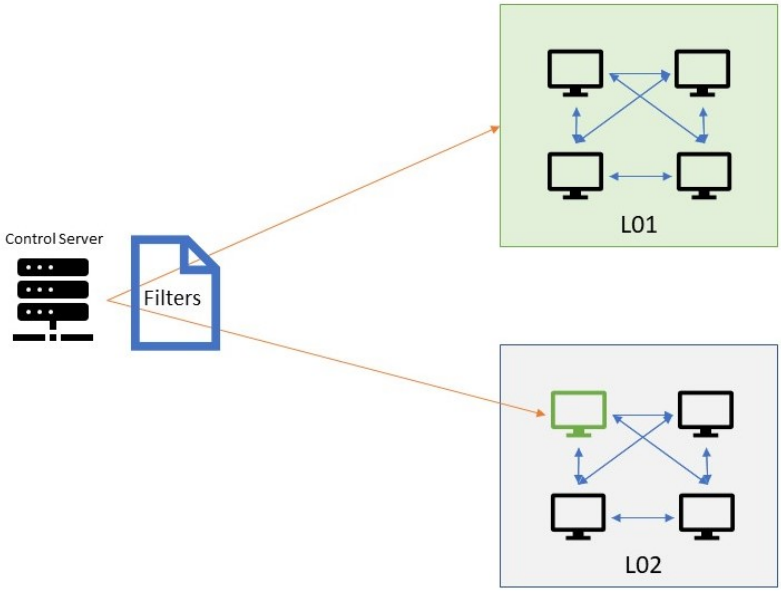

Figure 2.2: Conceptual deployment of filters to a node or set of nodes.

As it can be observed in the Figure 2.3, not all the laboratories have the same space, which will directly affect the number of computers that we can find inside. With Ansible, the number of computers inside a laboratory is not a problem. If one of the computer laboratories is checked, it will be similar to Figure 2.4, where the highlighted computer is the one in which the teacher will always be located.

The first concern of the system is how the server will be able to connect to all the computers in the laboratory. Ansible will be used for the automation of the deployment, the maintenance, and the take back of the filters in the nodes. To automatize these tasks, Ansible uses a technology called

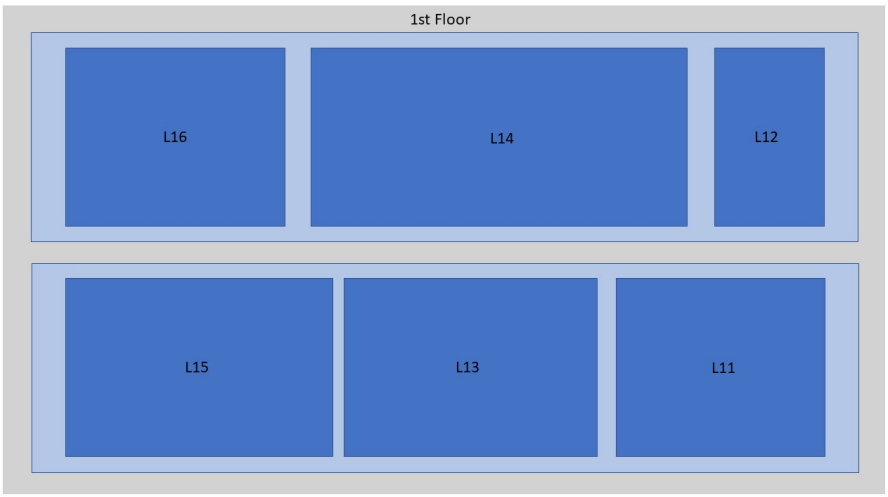

Figure 2.3: Layout of the first floor of EPSI.

'Playbooks'. Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process (*Ansible Official Documentation*, 2019). In order to perform all the tasks of the playbook, the server will become the user 'ansible'. This user has administrator privileges that will allow the server to perform all the tasks in the playbook in all of the machines demanded by the teacher.

After discussing how the system is going to work, now it is necessary to establish which filters are going to be available for the teachers or system administrators to apply. The most common requirement by the teachers right now is to ask for Internet connection limitations for the students. Thus, it is the most important filter that our application has to be able to apply. In order to apply this kind of filters it is essential to modify the iptables of the computer. With the iptables we are able to offer to a teacher different filters for an exam situation. This filters will be the next:

- UA Network
- Virtual Campus, Moodle UA and Web UA
- DLSI Servers
- EPS Servers
- Complete Restriction

In case the teacher wants to specify several independent URLs, the application will recommend to the teacher to contact the system administrator in order to check that there will be no problem accessing those URL during the exam. Also, as backup, the network filters applied in the iptables of each computer will be applied to the router through ACLs to provide a safety net in case a student

Figure 2.4: Layout of computer laboratory L01 from the school.

modifies the iptables as it is done right now. This can only be done in the building EPS1 because of the type of routers that can be found on that building. In addition to the network filters, the teacher will also be able to establish several system filters, for example, not allowing the usage of the USB memories or not allowing the execution of specific applications.

2.2.1.1. Implementation of Ansible Server

So far, the objectives of the system are clear. Now the document will focus on the Ansible server, which will be the responsible for all the automation in the system. The main objective of implementing the tasks through Ansible is to create a template. Through this template, every user will be able to configure the scenario for the laboratories from the school.

In order to implement that template, it is necessary that every task performed by the playbook is suitable for all computers. This can only be achieved by establishing a high level variable environment in which almost every single parameter that can be sent to a playbook. As a consequence of this environment, each different combination of values will have a different output once it is run through the template.

For the purpose of creating a high level variable environment the Ansible server will be structured as shown in Figure 2.5.

As it is illustrated in Figure 2.5 there are several files and folders in the structure which will guarantee

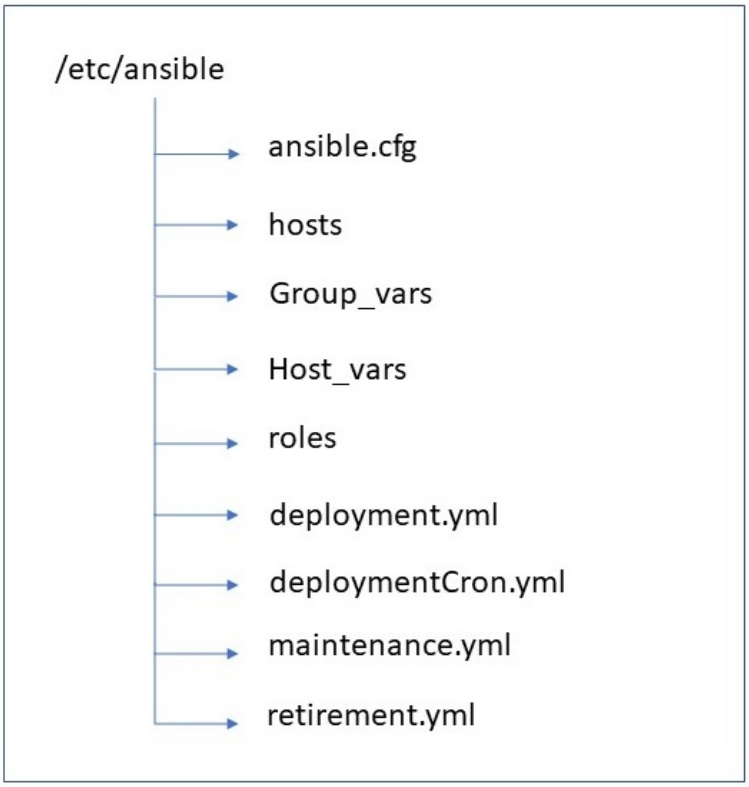

Figure 2.5: Structure of Ansible folder.

the reusability and flexibility of the final template. In order to do it, each of the folders and files has a different goal. Furthermore

On the first place, all the configuration parameters needed for the project can be set in the file "ansible.cfg". For example, to specify the SSH key that Ansible will use to connect to the nodes and the path to the inventory file the file should have the code shown in code 2.1

Code 2.1: Example of Ansible configuration file.

```
1 privatekeyfile = /home/centos/clubuntu.pem
2 inventory = /etc/ansible/hosts
```

On the second place, the file hosts is the inventory that Ansible will use to run the playbooks. In this file the name or IP addresses of the nodes that can be found in the system will be defined (Hochstein y Moser, 2017). This file is crucial to this project because in it the hosts can be divided by groups, which will allow ansible to run a playbook for an entire laboratory in just one command, avoiding to repeat the process for each of the computers in the laboratory. Also, in case that the target of the playbook are a set of hosts, the filters can be only applied to them if the teacher specifies it. In example, if a teacher wants to establish a filter for all the laboratories in the EPS1, instead of running the same playbook a number of times equivalent to the number of computers present in the building,

the structure in the file hosts shown in code 2.2 will provide Ansible the tool to execute a playbook one time for all the computers in the building, each one with its own different variables.

Code 2.2: Hosts file

```
[eps1:children]
Lab1
Lab2

[Lab1]
L01-1

[Lab2]
L02-1
```

On the third place, the folder group_vars will store two different types of files: The file all which will store the variables that can be used from every host in the system, and the other files will represent different groups. If a host belongs to a group, the variables specified in the group file will override the ones with the same name in the all file. This will allow the system to use the same template for all the hosts in the EPS in every possible situation. The group_vars folder in this project contains the files shown in Figure 2.6

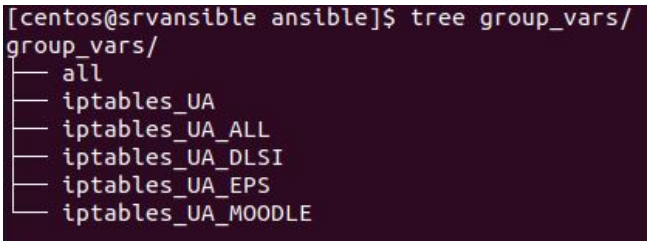

Figure 2.6: Content of the group_vars folder.

On the fourth place, the folder host_vars contains all the variables for each host. This variables will be applied over the group_vars in case the specific computer belongs to a group.

On the fifth place, in the folder roles will be defined each of the roles the system uses. For instance, in this project we will manage laboratories, so it has the role laboratories. A role consists in a container in which we can automatically load certain tasks, templates and handlers based on a known file structure. Grouping the content by roles also enhances the reusability of the playbooks. The laboratory will have three subfolders: handlers, templates and tasks as shown in Figure 2.7.

On sixth place, deployment.yml. This file is the playbook that will deploy the filters established by the teacher or SA to the laboratories. The file deployment.yml is shown in code A.2. This file will run the predefined tasks of the laboratory role, shown in code 2.3, to the hosts specified by the teacher.

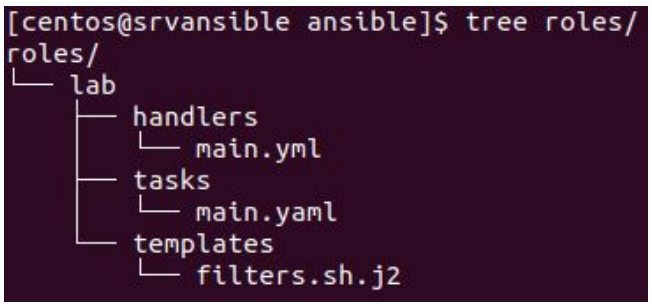

Figure 2.7: Content of the roles/laboratory/ folder.

Code 2.3: Deployment.yml file.

```
---
- name: Generate and Apply Configuration Files
  hosts: eps1
  roles:
   - laboratory
   vars_files:
     - group_vars/iptables
```

On seventh place the deploymentCron.yml. This file is complementary to the deployment playbook and will be in charge of setting the cronjob in order to execute the maintenance playbook periodically in the Ansible Server. The code of this file is shown in code 2.4. It is important to understand that in this playbook the hosts parameter will always be 127.0.0.1 which is equivalent to localhost, in other words, the ansible server itself. Also, the job that will do the cron established is the execution of the maintenance playbook every five minutes. This cronjobs will be stored in a crontab file. A crontab file contains instructions to the cron daemon of the general form: ”run this command at this time on this date” (*Linux Man*, 2019).

Code 2.4: DeploymentCron.yml file.

```
---
- name: Set Maintenance Playbook Cronjob
  hosts: 127.0.0.1
  become: true
  tasks:
  - name: Maintenance Cronjob
    cron:
      minute: "5"
      job: "ansible-playbook -i hosts -l Lab1 /etc/ansible/maintenance.yml"
```

In eighth place the maintenance.yml playbook. This file will be run every five minutes once the deployment playbook starts the process. The code of this file is shown in code 2.5. This playbook has two essential blocks. The first one is to execute the script getrules which will be inserted by the

deployment playbook in the previous phase in order to get the iptables filters at the time of the execution. Then, it will compare the filters that are currently active in the system with the ones that were set up in the deployment. If they are different, the task ”Execute Diff” will register the difference between both and a file called ”error.txt” will be generated. If it is generated it will be fetched by the Ansible server and stored.

Code 2.5: Maintenance.yml (1) file.

```
---
- name: Check filters
  hosts: eps1
  become: true
  tasks:
  - name: Execute Save Iptables Script
    shell: {{ ansible_user_dir }}/saveScript.sh

  - name: Execute Iptables Check
    command: {{ ansible_user_dir }}/getrules check

  - name: Execute Diff
    command: diff /etc/vim/checkNew.fw /etc/vim/checkOld.fw
    become: true
    failed_when: ”diff.rc > 1”
    register: diff

  - name: Generate Error File
    copy:
      content: ”{{ diff.stdout }}”
      dest: ”/etc/vim/error.txt”
    become: true
```

The second part of this playbook is shown in code 2.6 and will be only executed if a difference between the filters is found. As it can be appreciated, first, the server will get the error file. Then, it will send a mail with a report of the failures detected to the responsible of the filters, in other words, the person who asked for them. After contacting the responsible, the server will automatically apply again the filters established in the deployment to the node affected.

Code 2.6: Maintenance.yml (2) file.

```
  - name: Get Error file (if exists)
    fetch:
      src: /etc/vim/error.txt
      dest: /home/centos/report/error{{inventory_hostname}}.txt
      flat: true
    when: ”diff.stdout != ’’ ”
# mail to the responsible
```

```
8  - mail:
9      host: 127.0.0.1
10     subject: Ansible-Report
11     body: Hello, the system has found some irregularities in computer {{inventory_hostname}}, please check it.
12     to: "{{ responsible }}"
13     attach: /home/centos/report/error{{inventory_hostname}}.txt
14     charset: utf8
15   delegate_to: localhost
16   when: "diff.stdout != '' "
17 # deploy filters again
18  - name: Execute Configuration Files
19    command: sh {{ ansible_user_dir }}/filters.sh start
20    when: "diff.stdout != '' "
21
22  - name: Delete checkNewAux
23    file:
24      state: absent
25      path: "/etc/vim/checkNewAux.fw"
26    when: "diff.stdout != '' "
27
28  - name: Delete checkOld
29    file:
30      state: absent
31      path: "/etc/vim/checkOld.fw"
32    when: "diff.stdout != '' "
33
34  - name: Execute Save Iptables Script
35    command: sh {{ ansible_user_dir }}/saveScript.sh start
36    when: "diff.stdout != '' "
37
38  - name: Execute Iptables Check
39    command: {{ ansible_user_dir }}/getrules start
40    when: "diff.stdout != '' "
```

In ninth and last place the retirement.yml playbook. This file will be in charge of reverting the filters in every affected node, erase the files generated in this nodes and finally stop the cronjob in the ansible server to stop checking this filters. The code of this playbook is shown in code 2.7. There are more tasks in this playbook that the ones shown, but they all follow the same structure in order to delete the files generated during the deployment and the maintenance.

Code 2.7: Retirement.yml file.

```
1 ---
2 - name: Set filters back
3   hosts: eps1
4
5   tasks:
```

```
- name: Stop filters
  command: sh {{ ansible_user_dir }}/filters.sh stop
  become: true

- name: Delete Filters Script
  file:
    state: absent
    path: "{{ ansible_user_dir }}/filters.sh"
```

To sum up, "If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material." (*Ansible Official Documentation*, 2019)

2.2.1.2. Roles

An Ansible project can have many roles. Roles in Ansible allow you to group files together in a defined structure (Hall, 2015). In this project specifically it will have only one, which is laboratories and will be used to configure every computer laboratory from the university, regardless the building it is located in. As said previously, a role will have tasks, templates and handlers. The pillar of this system is the template. A template is a file which has a common part for all the scenarios and a series of variables whose value will depend on the hosts the playbook is applied to. The template that can be found in our laboratories role is in the file "template.sh.j2" and will generate a script that will be executed in the hosts specified by the playbook.

In the file "template.sh.j2" we can find some interesting details that will show how the automation is done. In code 2.8 is shown how the playbook will establish the cleaning rules, the default policies and the Ansible server access rules. This rules will be common to all the hosts, no matter where they are located at.

Code 2.8: Common Firewall Rules

```
case "$1" in start)
  # Cleaning rules and counters
  {% for rule in ipTablesCleaning -%}
    ($IPTABLES {{ rule }}) || Error $?
  {% endfor %}

  # Default policies
  {% for rule in ipTablesPolicy -%}
  ($IPTABLES {{ rule }}) || Error $?
  {% endfor %}

  ### Ansible access rule ###
  ($IPTABLES -A INPUT -p tcp --dport 22 -s {{ ansibleMaster }}/32 -j ACCEPT) || Error $?
  ($IPTABLES -A OUTPUT -p tcp -d {{ ansibleMaster }}/32 --sport 22 -j ACCEPT) || Error $?
```

After setting the common rules, the template will check if there is any iptable rule to be applied. If the user wants to apply any iptables rules it has to be defined in the group_vars directory first. If there is any rule to be applied, the process shown in 2.9 will be followed. This process consists in a loop which will execute the same process for all the defined rules in the group_vars file. This process consists into checking every possible attribute that an iptable rule can have in order to write it in the script that will be generated at the end of the execution.

Code 2.9: template.sh.j2 (1) file.

```
### Global Rules ###

{% if ipTablesRulesGlobal is defined -%}
  {% for globalRule in ipTablesRulesGlobal -%}
    {% set printRule = {'value': True} -%}
  ### Print rule ###
  {% if printRule.value and (globalRule.chain is defined or globalRule.table is defined or globalRule.oper is defined or
      ↪ globalRule.protocol is defined or globalRule.interfaceIN is defined or globalRule.interfaceOUT is defined or
      ↪ globalRule.state is defined or globalRule.saddr is defined or globalRule.sport is defined or globalRule.daddr
      ↪ is defined or globalRule.dport is defined or globalRule.target is defined or globalRule.free is defined or
      ↪ globalRule.comment is defined) -%}

  ### Comments ###
  {% if globalRule.commend is defined -%}
    # {{ rule.comment }}
  {% endif -%}

  ### Free rule ###
  {% if globalRule.free is defined -%}
    ($IPTABLES {{ globalRule.free }}) || Error $?
```

Moreover, once the common rules are set, the system will apply the specific rules of the group. A group is a set of variables defined for an specific purpose. An example of a group could be the one shown in Figure 2.8. If the system runs the deployment playbook with this group, it will apply all the rules defined in the group_vars file to all hosts or laboratories specified in the execution order.

In order to set the rules specified for a group in a file, the code shown in code 2.10 is needed. In it the system looks for all the rules of the group, and for each one it checks all the variables possible in order to create an iptable rule.

Code 2.10: template.sh.j2 (2) file.

```

            ($IPTABLES {{ "-t %s" %globalRule.table if defined else ""}}
            {{ "-%s" %globalRule.oper if globalRule.oper is defined else "-A" }}
            {{ globalRule.chain }}
            {{ "-p %s" %globalRule.protocol if globalRule.protocol is defined else "" }}
```

```
{{ "-i %s" %globalRule.interfaceIN if globalRule.interfaceIN is defined else "" }}
{{ "-o %s" %globalRule.interfaceOUT if globalRule.interfaceOUT is defined else "" }}
{{ "-d %s" %Raddr if Raddr!="NULL" else "" }}
{{ "--dport %s" %Rdport if Rdport!="NULL" else "" }}
{{ "-s %s" %Rsaddr if Rsaddr!="NULL" else "" }}
{{ "--sport %s" %Rsport if Rsport!="NULL" else "" }}
{{ "-m state --state %s" %globalRule.state|join(',') if globalRule.state is defined else "" }}
-j {{ Rtarget }}) || Error $?
```

```
[centos@srvansible ansible]$ cat group_vars/iptables
ipTablesRulesGlobal:
- label: localhost access
  chain: IO
  interfaceIN: lo
  interfaceOUT: lo
  saddr: [ 127.0.0.1 ]
  comment: localhost access

- label: DHCP access
  chain: IO
  protocol: udp
  dport: [ 67 ]
  comment: DHCP port (67) access

- label: SSH access from Admon subnet
  chain: IO
  table: filter
  oper: I
  protocol: tcp
  interfaceIN: eth0
  interfaceOUT: eth0
  state: [ NEW, ESTABLISHED ]
  saddr: [ 172.20.40.0/24 ]
  daddr: [ 0.0.0.0/32 ]
  dport: [ 22 ]
  target: [ ACCEPT ]
  comment: "SSH access from Admon subnet"
```

Figure 2.8: Group vars example.

After creating the new file with the template and the selected variables, the tasks of the role will be executed. This tasks can be found in the ansible folder in '/roles/laboratories/tasks/main.yml'. The tasks performed by this roles are the ones shown in code 2.11. In this code we can find several tasks. The fist one will make a script with the rules to be applied and it will send it to the node. The second task will start the script and set the filters. The third one will copy the executable file of a program created to save the current iptables of a system in a specific location .The fourth one will save the current iptables in order to be able to check if the filters have been altered later. The fifth one will copy the executable file of another program created to set the scenario to be able to compare the present rules with the ones established in the first place. Furthermore, it is important to highlight that all the files sent to the nodes will have "mode=0700" which indicates the permissions of the files. In this case only the owner, which will be the root, can have access to this files.

Code 2.11: /roles/laboratories/tasks/main.yml.

```
---

- name: Generate Configuration Files
  template: src=/etc/ansible/roles/lab/templates/filters.sh.j2 dest={{ ansible_user_dir }}/filters.sh owner=ubuntu mode
      =0700

- name: Execute Configuration Files
  command: sh {{ ansible_user_dir }}/filters.sh start
  become: true

- name: Save-iptables
  copy:
    src: /home/centos/saveScript.sh
    dest: /{{ ansible_user_dir }}/saveScript.sh
    owner: ubuntu
    mode: 0700

- name: Execute Save Iptables Script
  command: sh {{ ansible_user_dir }}/saveScript.sh start
  become: true

- name: Insert iptablesCheck program
  copy:
    src: /home/centos/program/maintenance/getrules
    dest: /{{ ansible_user_dir }}/getrules
    owner: ubuntu
    mode: 0700

- name: Execute Iptables Check
  command: /{{ ansible_user_dir }}/getrules start
  become: true
```

2.2.2. Server automation

In order to automatize the entire process, it is necessary to design an application that operates as an intermediary between the users and the automation system. Since there are two types of users, teachers and system administrators, the program will allow the user to choose as shown in Figure 2.9. If the selected option in Figure 2.9 is 1, the program will start asking for the data needed in order to

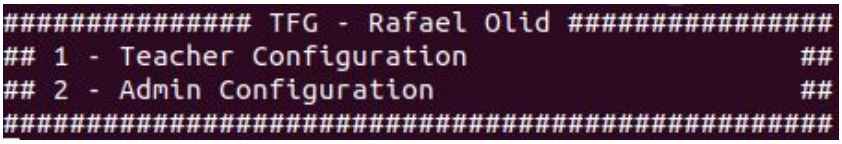

Figure 2.9: Initial menu of the application.

schedule the necessary tasks to apply the deployment playbook. The data inserted by the user in the app, as illustrated in Figure 2.10, will be used as parameters for the playbook and also sent to a log in order to keep track of the activities performed by the system. In order to apply the filters specified

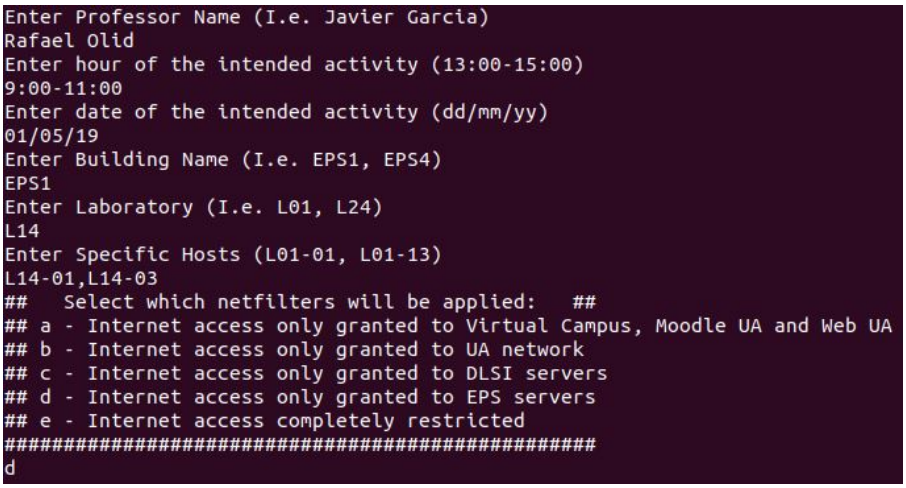

Figure 2.10: Example of filled form of the application.

by the user, our application has to modify two templates that will be filled with the parameters to generate two playbooks. The first generated playbook will be the one to establish all the filters in the laboratory. The second playbook will establish a cronjob to execute the maintenance playbook during the exam. What the program will modify in this file is the variable files that the role will use later. As shown before, there is a group of variables for every netfilter option, and only one of this files can be at the same time in the playbook to work.

2.3. System Hardening

It should have been noticed that this entire system works through SSH. This makes the server a critical point for the system because, if someone manages to access the server, they will be able to access every node defined for the Ansible system. In order to provide a higher level of security for the most important component of the system, I have decided to harden the SSH service of the system with Multifactor Authentication and Port Knocking.

2.3.1. 2FA with Google Authenticator

SSH uses passwords for authentication by default, and most SSH hardening instructions recommend using an SSH key instead, but his is still only a single authentication factor. With the objective of increasing the security of the server, a Two Multifactor Authentication (2FA) has been set up. With this two factor authentication, every time a user wants to access the server it will not only be enough

with a user and a password, it will also be necessary a code as shown in figure 2.11. Moreover, this second authentication factor will change every thirty seconds. In other words, a bad actor would have to compromise the computer and the mobile phone to get in. The verification code asked will be

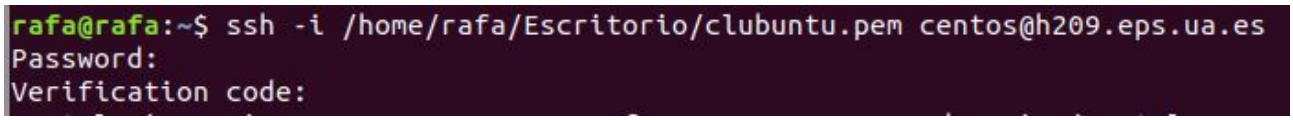

Figure 2.11: SSH login command.

the one indicated in the Google Authenticator app in the system administrator phone, an example is shown in figure 2.12. A more detailed explanation of 2FA and Google Authenticator is located in

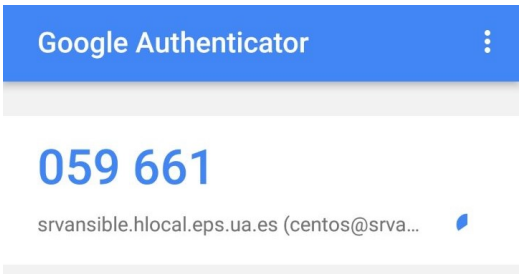

Figure 2.12: Google Authenticator Mobile Applicaiton.

Appendix A.

2.3.2. Port Knocking

There is software developed, such as knockd, that allows us to use a very interesting protection technique: Port Knocking. The idea is that, when there is a specific sequence of connections to certain ports, another is enabled, which is our ultimate goal. For example, we have disabled access to the SSH server and, after receiving a connection sequence to ports 11021, 11014 and 11019, the connection is activated, for that IP of origin, to port 22 of the SSH. It is one more technique used for SSH fortification. After the set up of the port knocking on the server, an ssh connection will be established as shown in figure 2.13. A more detailed explanation of Port Knocking can be found in Appendix A.

2.4. System Instantiating

As said previously in chapter 1.4, the objective of this work was to implement an automated system to manage the access of a group of computers to network and system resources. After the implementation done in this chapter the system is the one shown in Figure 2.14. As it can be observed the application and the automation system have been gathered in a server from which the system will be

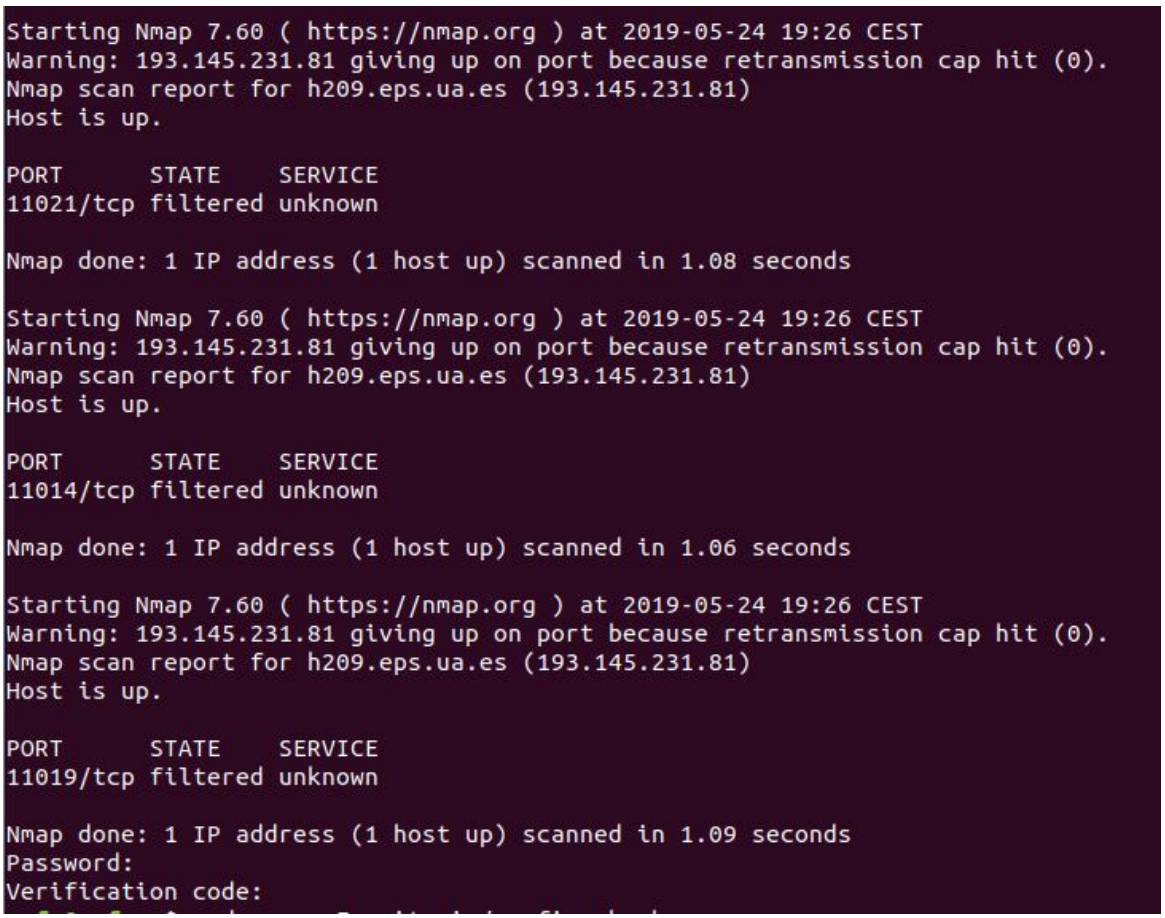

Figure 2.13: SSH login with Port Knocking.

able to apply the filters required by the teacher or the SA in any node or group of nodes from the laboratories of the EPS.

After analyzing the main structure of an Ansible project, now we will explain the three different phases of this project. The first one is the deployment. This phase consists in the deployment of the initial playbook. This playbook will contain the necessary tasks in order to generate the configuration files for each of the nodes the teacher or SA asked and run this files in the nodes. The second one is the maintenance. This second phase also consists in the periodically execution of a playbook. The objective of this playbook is to check if the filters applied are still running in each of the nodes. If the tasks are not accomplished it will send a mail to the teacher. Finally, the third phase, retreat. This playbook will be in charge of eliminating the filters in the nodes and to stop the support phase in the server.

In order to start the first phase, when teacher or the system administrator fulfill the form, the content of the form will be used to run a program in the Ansible server that will set an 'at' command which will run the first Ansible playbook when the date specified by the teacher arrives. As said before, the first Ansible playbook will deploy the configuration to all the nodes, but also it will set a crontab

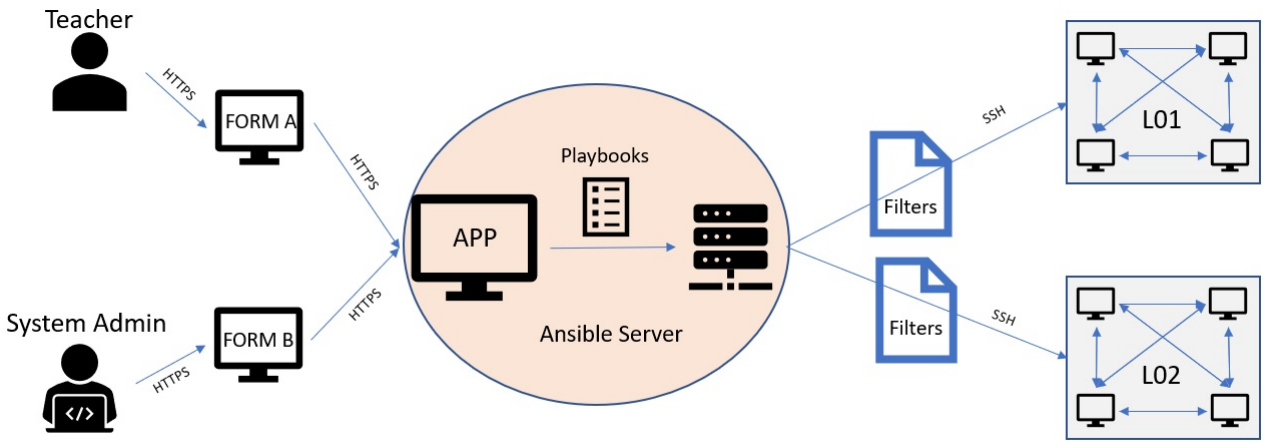

Figure 2.14: Components of the implemented system.

and a second 'at' command. The crontab will set the execution of the second playbook with the same parameters as it was done on the first one repeatedly every five minutes. The second 'at' command will set the execution of the last playbook, the one needed to restore the filters.

3. Validation

In this chapter, I am going to perform several tests to evaluate the performance. These tests will simulate different situations, including normal functioning, which means a network access filtering without any impediment; and various malfunctioning cases. Furthermore, in order to perform the following tests, the system and the nodes have been prepared according to the requirements made in the previous chapters to ensure that the project can replace the current system.

3.1. Normal Functioning

To validate the system, the first test will evaluate the ideal execution. An ideal execution refers to a situation where the filters are applied and neither the network hardware fails nor the students break the limitations. With this test I want to ensure the correct functioning of the system. An ideal execution must follow the next steps without any inconvenience:

- Teacher or SA fills the form in the application.
- The application generates correctly the playbooks.
- The filters are deployed properly in the nodes specified in the form.
- The maintenance playbook runs every 5 minutes to check the filters, and the filters are not altered in the node.
- The retirement playbook is executed at the end of the exam.

To perform this test, a form is filled as a teacher in the application implemented in the Ansible server. The result of that form is shown in figure 3.1. All the information asked in the form will not only be used to schedule the execution of the tasks and the variables selected, it will also be stored in a log file in case the SA needs to consult anything afterwards. The log file is shown in figure 3.2.

In order to help the reader understand what is happening at every step of the process, the execution of the playbooks will be done manually to show which tasks are being executed. As shown in figure 3.1 the test is going to be performed in L01, a laboratory composed in my virtual environment by the

```
############## Professor Configuration  ###############
## Professor: ror10@alu.ua.es
## Day Scheduled: 29/05/19
## Time Scheduled: 9:00-11:00
## Building: eps1
## Laboratory: L01
## Specific computers: none
## Netfilters: Internet access completely restricted
## Hardware filters: not implemented yet
#####################################################
```

Figure 3.1: Output of the application after filling the form.

```
diego@ua.es | 22/05/19 | 19:00-20:00 | eps1 | L01 | none | Internet access completely restricted
martin@ua.es | 23/05/19 | 16:00-16:44 | eps1 | L01 | none | Internet access completely restricted
ror10@alu.ua.es | 29/05/19 | 9:00-11:00 | eps1 | L01 | none | Internet access completely restricted
```

Figure 3.2: Examples of log file from the application.

computer L01-1. First, I will execute the deployment playbook, getting the output shown in figure 3.3. As it can be seen, the tasks specified in the playbook were executed and the deployment was successfully done.

```
[centos@srvansible ansible]$ ansible-playbook -i hosts -l L01 deploymentL01.yml

PLAY [Generate Configuration Files] ********************************************
*******

TASK [Gathering Facts] *********************************************************
*******
ok: [L01-1]

TASK [lab : Generate Configuration Files] **************************************
*******
changed: [L01-1]

TASK [lab : Execute Configuration Files] ***************************************
*******
changed: [L01-1]

TASK [lab : Save-iptables] *****************************************************
*******
changed: [L01-1]

TASK [lab : Execute Save Iptables Script] **************************************
*******
changed: [L01-1]

TASK [lab : Insert iptablesCheck program] **************************************
*******
changed: [L01-1]

TASK [lab : Execute Iptables Check] ********************************************
*******
changed: [L01-1]

PLAY RECAP *********************************************************************
*******
L01-1                      : ok=7    changed=6    unreachable=0    failed=0
```

Figure 3.3: Output of the deployment playbook execution.

After the deployment of the filters, the system will program a cronjob to execute the maintenance playbook every five minutes and also it will schedule the execution of the retirement playbook at the ending time of the exam. The execution of this playbook, if every node behaves as it should, without

any system failure or user interference, is the one shown in figure 3.4 where the check of the filters is done. As shown in the figure, there are a several tasks which are skipped, this is because the system has not found an error, so it does not have to execute the tasks which will inform the teacher or SA who asked for the filters.

```
[centos@srvansible ansible]$ ansible-playbook -i hosts -l L01 maintenance.yml

PLAY [Check filters] ***********************************************************

TASK [Gathering Facts] *********************************************************
ok: [L01-1]

TASK [Execute Save Iptables Script] ********************************************
changed: [L01-1]

TASK [Execute Iptables Check] **************************************************
changed: [L01-1]

TASK [Execute Diff] ************************************************************
changed: [L01-1]

TASK [Generate Error File] *****************************************************
changed: [L01-1]

TASK [Get Error file (if exists)] **********************************************
skipping: [L01-1]

TASK [Check if the file exists] ************************************************
skipping: [L01-1]

TASK [mail] ********************************************************************
skipping: [L01-1]

TASK [Execute Configuration Files] *********************************************
skipping: [L01-1]

TASK [Delete checkNewAux] ******************************************************
skipping: [L01-1]

TASK [Delete checkOld] *********************************************************
skipping: [L01-1]

TASK [Execute Save Iptables Script] ********************************************
skipping: [L01-1]

TASK [Execute Iptables Check] **************************************************
skipping: [L01-1]

PLAY RECAP *********************************************************************
L01-1                      : ok=5    changed=4    unreachable=0    failed=0
```

Figure 3.4: Output of the maintenance playbook execution.

To finish the process, the execution of the retreat playbook will be done as shown in figure 3.5. This will set back the filters and delete all the files generated by the previous playbooks, leaving the node exactly as it was in the first place.

```
[centos@srvansible ansible]$ ansible-playbook -i hosts -l L01 retreat.yml

PLAY [Set filters back] ********************************************************

TASK [Gathering Facts] *********************************************************
ok: [L01-1]

TASK [Stop filters] ************************************************************
changed: [L01-1]

TASK [Delete Filters Script] ***************************************************
changed: [L01-1]

TASK [Delete saveScript] *******************************************************
changed: [L01-1]

TASK [Delete getRules] *********************************************************
changed: [L01-1]

TASK [Delete checkNewAux] ******************************************************
changed: [L01-1]

TASK [Delete checkNew] *********************************************************
changed: [L01-1]

TASK [Delete checkOld] *********************************************************
changed: [L01-1]

TASK [Check error file exists] *************************************************
ok: [L01-1]

TASK [Delete error file] *******************************************************
changed: [L01-1]

PLAY RECAP *********************************************************************
L01-1                      : ok=10   changed=8    unreachable=0    failed=0
```

Figure 3.5: Output of the retirement playbook execution.

After the last playbook was applied, if the iptables are checked in the nodes from the laboratory they will be the ones shown in Figure 3.6

```
ubuntu@cllab101:~$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD DROP
-P OUTPUT ACCEPT
```

Figure 3.6: Net filters of the nodes after retreat playbook execution.

3.2. System Failures

A system failure is an error which is originated in the server. There are different types of system failures: The first type of system failure is produced if the server is turned down. In this case, even though the server is down, the filters will be still applied in the nodes, and when the server is up again, it will run the maintenance playbook as established before, checking the filters again. The second type of system failure is the one produced when the SSH connection cannot been established to the node. Due to Ansible's functioning, it is not possible to automate this proccess. Therefore, when the system programs the execution of the playbooks, it will be done as shown in code 3.1, sending the output of the command to a log, so the system administrator can observe the cause of the error in the system.

Code 3.1: Playbook execution command.

```
1 ansible-playbook -i hosts -l L01 deployment >> /home/centos/log/logL01-13:00
```

For instance, if the deployment playbook is run for a computer and the server cannot establish an SSH session with it, the output will be one shown in Figure 3.7

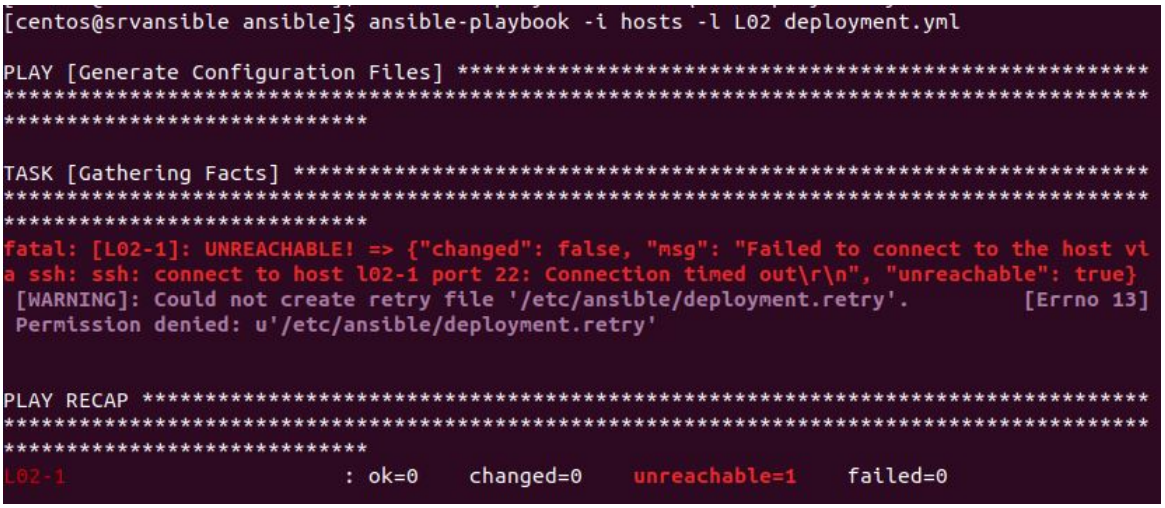

Figure 3.7: Output of a playbook when the SSH connection cannot be established.

3.3. User Interference

In this section, I will show how the system will answer to any interference caused by a user in a node. At this moment, the implemented system apply the filters via iptables. In order to see what happens when a user changes the rules of the iptables we will change the iptables policies and execute the maintenance playbook. This simulation will be exactly the same as the one done in section 3.1, where the deployment of the filters has been done and the maintenance playbook has been executed

one time. Now that the maintenance playbook has been executed, the user has a small window (it is five minutes but the SA can change it in order to make the checks faster) to revoke the filters. It is important to clarify that in this validation case the only possibility of modifying the filters is if the user manages to log as root, which can only be done if the user knows the password or if the user has an exploit. Once they get the root permissions they will be able to set a different kind of filters as the ones shown in 3.8, where they have changed the policies INPUT and OUTPUT, allowing the node to communicate with other servers or machines.

```
ubuntu@cllab101:~$ sudo iptables -P INPUT ACCEPT
ubuntu@cllab101:~$ sudo iptables -P OUTPUT ACCEPT
ubuntu@cllab101:~$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD DROP
-P OUTPUT ACCEPT
```

Figure 3.8: Example of unauthorized modification.

Once the user has changed the iptables, when the next execution of the maintenance playbook is done the output of it will be the one shown in 3.9, where there are not skipped tasks this time due to the differences find between the current filters and the ones from the deployment.

When the maintenance playbook is executed in this case, the responsible will receive a mail with the information of the node or nodes that showed an unusual behaviour. This mail will look like the one shown in 3.10, where the node is indicated and a file with the difference is attached.

After the maintenance playbook executions, the filters will be set back as established by the responsible in the first place and next time the playbook is executed it will get a normal output, as shown in the normal functioning before.

```
[centos@srvansible ansible]$ ansible-playbook -i hosts -l L01 maintenance.yml

PLAY [Check filters] ***********************************************************

TASK [Gathering Facts] *********************************************************
ok: [L01-1]

TASK [Execute Save Iptables Script] ********************************************
changed: [L01-1]

TASK [Execute Iptables Check] **************************************************
changed: [L01-1]

TASK [Execute Diff] ************************************************************
changed: [L01-1]

TASK [Generate Error File] *****************************************************
changed: [L01-1]

TASK [Get Error file (if exists)] **********************************************
changed: [L01-1]

TASK [Check if the file exists] ************************************************
ok: [L01-1]

TASK [mail] ********************************************************************
ok: [L01-1 -> localhost]

TASK [Execute Configuration Files] *********************************************
changed: [L01-1]

TASK [Delete checkNewAux] ******************************************************
changed: [L01-1]

TASK [Delete checkOld] *********************************************************
changed: [L01-1]

TASK [Execute Save Iptables Script] ********************************************
changed: [L01-1]

TASK [Execute Iptables Check] **************************************************
changed: [L01-1]

PLAY RECAP *********************************************************************
L01-1                      : ok=13   changed=10   unreachable=0    failed=0
```

Figure 3.9: Output of the retreat playbook execution.

Figure 3.10: Mail sent to the responsible when unusual behaviour is found in a node.

4. Conclusion

This last chapter contains the final thoughts and conclusions about this work. First a summary of the main conclusions of this work will be presented. Later,I will comment future lines in which this project can continue. Finally, I will share my thoughts about this project.

4.1. Conclusions

In this bachelor thesis, I have proposed an automated system for administration delegation to manage the access of a group of computers to network resources. Taking into account the tool that is currently used in the EPS, the aim was to migrate it into a new tool which is not limited by the network level in which the filters are applied, but only by the operating system of the nodes.

In this project, I analyzed the current system used by the EPS to determine the existing problems that have to be dealt with. The implemented system uses Ansible's automation technology to deploy the filters through all the laboratories from the EPS, and it will only be limited by the operating system from the computers. Moreover, I will like to highlight that, with this new system, the filters can be applied not only in one of the buildings (EPS1), but in all of them. Furthermore, the implemented system has been provided with tools to inform the teacher if a student surpasses the filters. Lastly, several validation tests have been performed in the system in order to replicate different real situations, in which the system will be applied once deployed.

4.2. Future Lines

The implementation carried out in this project covers part of the general proposal. Therefore, there are different extension lines which can be done.

In the short term, I propose to develop a more user-friendly interface. For instance, to extend the current website while preserving the command line version developed in this project.

As long term future lines, the addition of new system functionalities is proposed. The fact of being able to control what the user is able to do with every application in the node could be a potential improvement, for example, by using AppArmor to establish system filters.

Finally, it is also proposed to migrate all the developed technology (designed for Linux systems) to Windows, in order to generalize it for other operative system.

4.3. Personal Thoughts

Personally, I think this bachelor thesis has given me the chance to acquire knowledge that it is not taught in the degree. It helped me to improve my skills to face larger projects, and particularly, the specific knowledge and tools I used and learned. I am really glad that I have taken the first steps into the automation world, I am confident that the skills developed during the implementation of this system will be useful in my future as engineer.

Bibliography

Alfke, M. (2017). *Puppet 5 Essentials: A fast-paced guide to automating your infrastructure* (Third ed.).

Alibi, M. (2018). *Ansible Quick Start Guide: Control and monitor infrastructures of any size, physical or virtual* (First ed.).

Álvarez Martín, C., y Pérez González, P. (2017). *Hardening de servidores GNU/Linux* (3rd ed.).

Ansible Official Documentation. (2019). Descargado 2019-05-27, de `https://docs.ansible.com/`

Bass, L., Weber, I., y Zhu, L. (2015). *DevOps: A Software Architect's Perspective.*

Hall, D. (2015). *Ansible Configuration Management* (Second ed.).

Hochstein, L., y Moser, R. (2017). *Ansible: Up and Running.*

Keif Morris. (2016). *Infrastructure as Code: Managing Servers in the Cloud* (1st ed.).

Linux Man. (2019). Descargado 2019-04-19, de `https://linux.die.net/man/5/crontab`

Pieplu, R. (2019). Ground control segment automated deployment and configuration with ansible and git. En *2018 spaceops conference.* doi: 10.2514/6.2018-2337

Rash, M. (2007). *Linux Firewalls Attack Detection and Response with iptables.*

Shah, G. (2015). *Ansible Playbook Essentials: Design automation blueprints using Ansible's playbooks to orchestrate and manage your multitier infrastructure.*

Taylor, M., y Vargo, S. (2013). *Learning Chef: A Guide to Configuration Management and Automation* (First ed.).

University of Alicante. (2019). Descargado 2019-04-19, de `https://www.ua.es`

List of Acronyms and Abbreviations

2FA Two-factor Authentication.

EPS Superior Polytechnic School of University of Alicante.

EPS1 First building of Superior Polytechnic School.

L01 Laboratory One.

L01-1 Computer One from laboratory one.

MFA Multifactor Authentication.

SA System Administrator.

SSH Secure Shell.

A. SSH Hardening

A.1. 2FA and Google Authenticator Set Up

As said in the implementation chapter, the server has been provided of a 2FA set up in order to increase the security. The steps followed to set this hardening up have been the following ones:

- Install Google Authenticator in our phone
- Install and configure Google PAM
- Configuring Open SSH

PAM, which stands for Pluggable Authentication Module, is an authentication infrastructure used on Linux systems to authenticate a user. First, we need to add the EPEL (Extra Packages for Enterprise Linux) repo.

Code A.1: Command to Add EPEL

```
sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

Next, install the PAM.

Code A.2: Command to Install PAM

```
sudo yum install google-authenticator
```

With the PAM installed, we'll use a helper app that comes with the PAM to generate a TOTP key for the user you want to add a second factor to. This key is generated on a user-by-user basis, not system-wide. Run the initialization app.

Code A.3: Command to Install PAM

```
google-authenticator
```

After we run the command, we'll be asked a few questions:

1. With the first one, we are indicating that we want the tokens to be based on time and to update the HOME/.google_authenticator file which is where you will store important information such as the link and the code mentioned in the previous point.

2. The second asks if we want to disable the use of the same token by different users
3. The third indicates the default time (30) to compensate for possible temporary differences between client and server.
4. And in the latter, limits the number of attempts to 3 each 30 to avoid brute force attacks.

After answering this questions, we will have to scan the QR code in order to receive a six number code in our mobile application which will change every 30 seconds. Now, we will configure OpenSSH. First we will open sshd configuration file:

Code A.4: Open configuration file.

```
sudo nano /etc/pam.d/sshd
```

Then, we will add the following line at the bottom of the file:

Code A.5: Modification of configuration file.

```
auth required pam_google_authenticator.so nullok
```

This allows users without a OATH-TOTP token to still log in using their SSH key. Next, we'll configure SSH to support this kind of authentication. Open the SSH configuration file for editing.

Code A.6: Open sshd configuration file.

```
sudo nano /etc/ssh/sshd_config
```

The affected lines are the following:

Code A.7: Configuration of sshd file.

```
ChallengeResponseAuthentication yes
#ChallengeResponseAuthentication no
```

And finally, restart the service:

Code A.8: Command to restart sshd service

```
sudo systemctl restart sshd.service
```

A.2. Port Knocking

As said in Chapter 2, Port Knocking is a very interesting protection technique. The idea is that, when there is a specific sequence of connections to certain ports, another is enabled, which is our ultimate goal (Álvarez Martín y Pérez González, 2017). For example, we have disabled access to the SSH server and, after receiving a connection sequence to ports 11021, 11014 and 11019, the connection

is activated, for that IP of origin, to port 22 of the SSH. This function can be developed in a very simple way with iptables and the recent module (that manages dynamic lists of IPs and that we already use to control the number of incoming connections from the same IP in this entry). For example, for the SSH activation sequence to be: 4000, 2000, 3000, we should configure the iptables as follows:

Code A.9: Iptables commands to establish port knocking.

```
1 /sbin/iptables -N CHAIN_STEP2 /sbin/iptables -A CHAIN_STEP2 -m recent --name STEP1 --remove /sbin/↩
      ↪ iptables -A CHAIN_STEP2 -m recent --name STEP2 --set /sbin/iptables -A CHAIN_STEP2 -j LOG ↩
      ↪ --log-prefix "ENTERING STEP 2: "
2
3 /sbin/iptables -N CHAIN_STEP3 /sbin/iptables -A CHAIN_STEP3 -m recent --name STEP2 --remove /sbin/↩
      ↪ iptables -A CHAIN_STEP3 -m recent --name STEP3 --set /sbin/iptables -A CHAIN_STEP3 -j LOG ↩
      ↪ --log-prefix "ENTERING STEP 3: "
4
5 /sbin/iptables -I INPUT -m recent --update --name STEP1 /sbin/iptables -I INPUT -p tcp --dport 4000 -m ↩
      ↪ recent --set --name PASO1 /sbin/iptables -I INPUT -p tcp --dport 2000 -m recent --rcheck --name ↩
      ↪ STEP1 -j CHAIN_STEP2 /sbin/iptables -I INPUT -p tcp --dport 3000 -m recent --rcheck --name ↩
      ↪ STEP2 -j CHAIN_STEP3
6
7 /sbin/iptables -I INPUT -p tcp --dport 22 -m recent --rcheck --seconds 5 --name STEP3 -j ACCEPT
```

These lines that implement the port knocking functionality must be added to the filters that we have in our server and we must take into account that:

1. Communications for SSH must be DISABLED. With, for example, /sbin/iptables -P INPUT DROP we achieve this. The activation is done with the port knocking commands. If we don't want to be so drastic: /sbin/iptables -A INPUT -p tcp -dport 22 -j DROP

2. We must have a rule that enables established connections. If not, our session will last 5 . For example, one could be: /sbin/iptables -A INPUT -m state -state RELATED,ESTABLISHED -j ACCEPT

B. Development Tools

In this project I will use several tools to keep track of the progress done and also for the implementation of the system. This tools can be divided in organization and support tools.

B.1. Organization Tools

During the development of this project, I mainly had to work with my advisors. In order to do that, we had to set an environment in which they could always know the situation of the project and point something out if they considered it needed a correction or a different approachment. In order to do that, we decided to use three different tools to keep all the project organized and prevent any kind of unexpected outcomes.

- Trello. We used trello in order to organize the work in objectives and with priorities. We have used this tool after every meeting in order to setup the objectives for the next ones or the future tasks that I will have.
- Google Drive. In order to store all the materials from the project, like source code, articles related, images and ideas.
- Overleaf. We also used this online text editor in order to be able to access to the document I was developing.

B.2. Support Tools

As a consequence of applying the filters directly to the nodes, and not to the network, we can take advantage of some tools that can be used in GNU/Linux systems. The project will be supported by two main tools, Iptables and AppArmor. On the one hand iptables is an administration tool for IPv4 packet filtering and NAT. It is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. In it several tables can be found, each one contains a number of built-in chains and may also contain user-defined chains. Each chain is a list of rules which can match a set of

packets (Rash, 2007). Each rule specifies what to do with a packet that matches. If the packet does not match, the next rule in the chain is the examined; if it does match, then the next rule is specified by the value of the target, which can be the name of a user-defined chain or one of the special values ACCEPT, DROP or RETURN. ACCEPT means to let the packet through. DROP means to drop the packet on the floor. RETURN means stop traversing this chain and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet. In Figure B.1 an example of iptables can be observed. In this case, that is the iptables that the Ansible server has inside a virtual scenario that has been created for the development phase of the project.

```
[centos@srvansible ansible]$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD DROP
-P OUTPUT ACCEPT
-A INPUT -s 127.0.0.1/32 -i lo -j ACCEPT
-A INPUT -s 127.0.1.1/32 -j ACCEPT
-A INPUT -p udp -m udp --sport 53 -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -d 127.0.0.1/32 -o lo -j ACCEPT
-A OUTPUT -d 127.0.1.1/32 -j ACCEPT
-A OUTPUT -p udp -m udp --dport 53 -j ACCEPT
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
[centos@srvansible ansible]$
```

Figure B.1: Iptables of AnsibleSRV.

In Figure B.1 several things can be highlighted. On the first place, the first three lines contain policies, which are the default answer for a packet if it does not match any of the defined lines.
On the other hand, to provide a better control of the nodes to the teacher we will use AppArmor in order to have a control not only in the network of the laboratory, also in the applications the node users will be able to execute. AppArmor is Mandatory Access Control (MAC) like security system for Linux. AppArmor confines individual programs to a set of files, capabilities, network access and limits, collectively known as the AppArmor policy for the program, or simply as a profile. New or modified policy can be applied to the running system without a reboot.